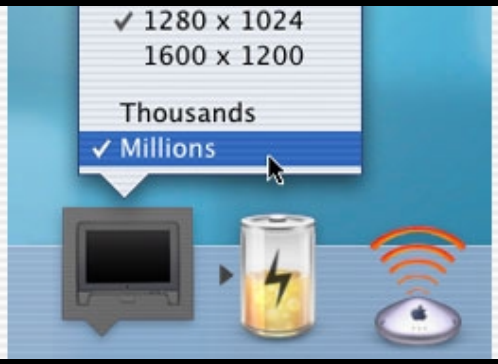
Session 104

I/O Kit Update



Craig Keithley
USB & FireWire Technology Manager
Apple Worldwide Developer Relations

Session 104

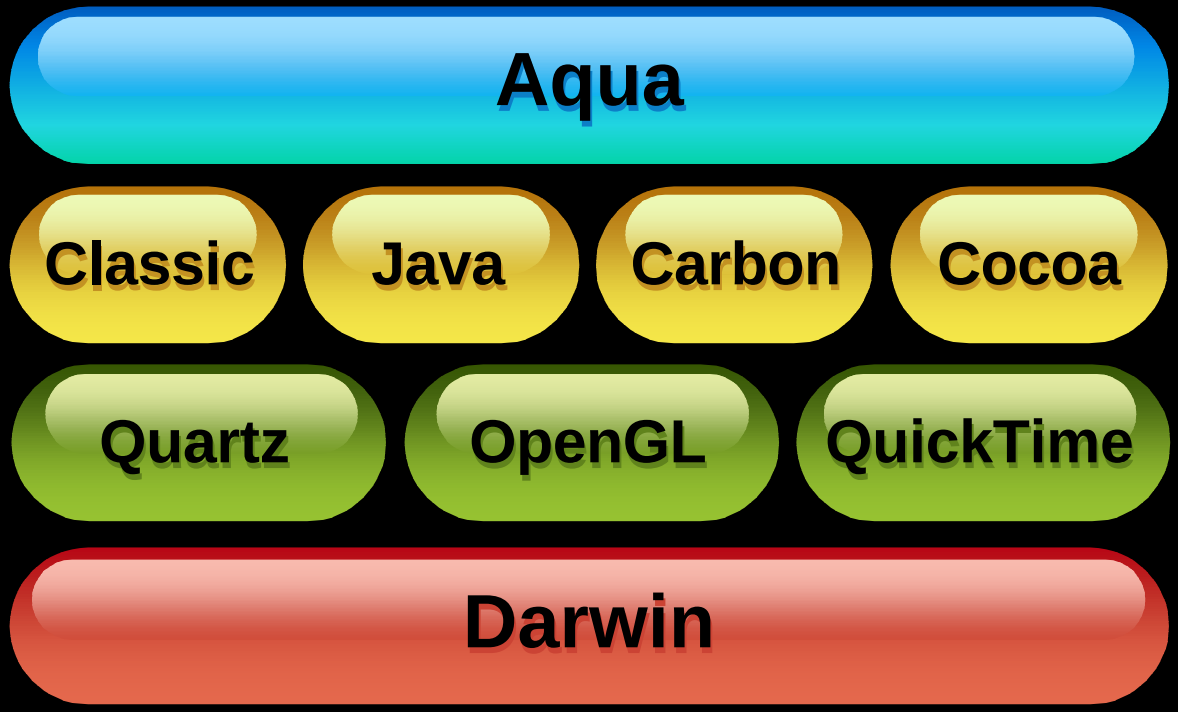
I/O Kit Update

Dean Reece
I/O Kit Manager

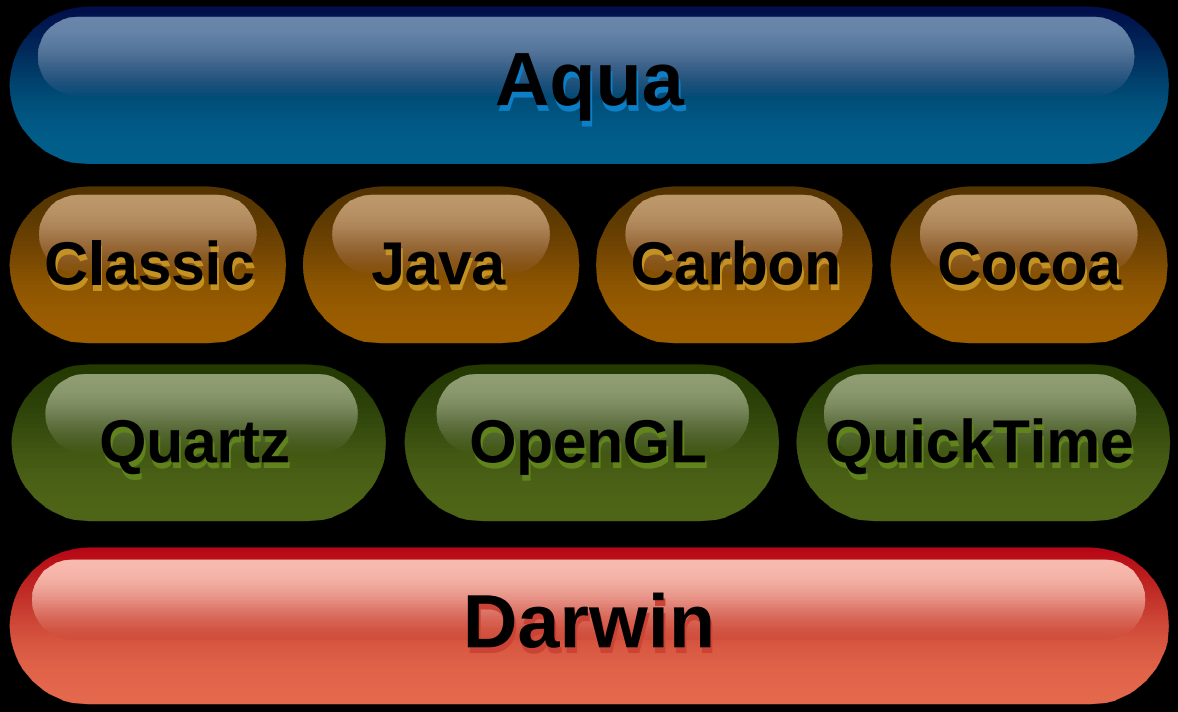
Introduction

- Design and scope of I/O Kit
- Kernel Extensions
- Application access to devices

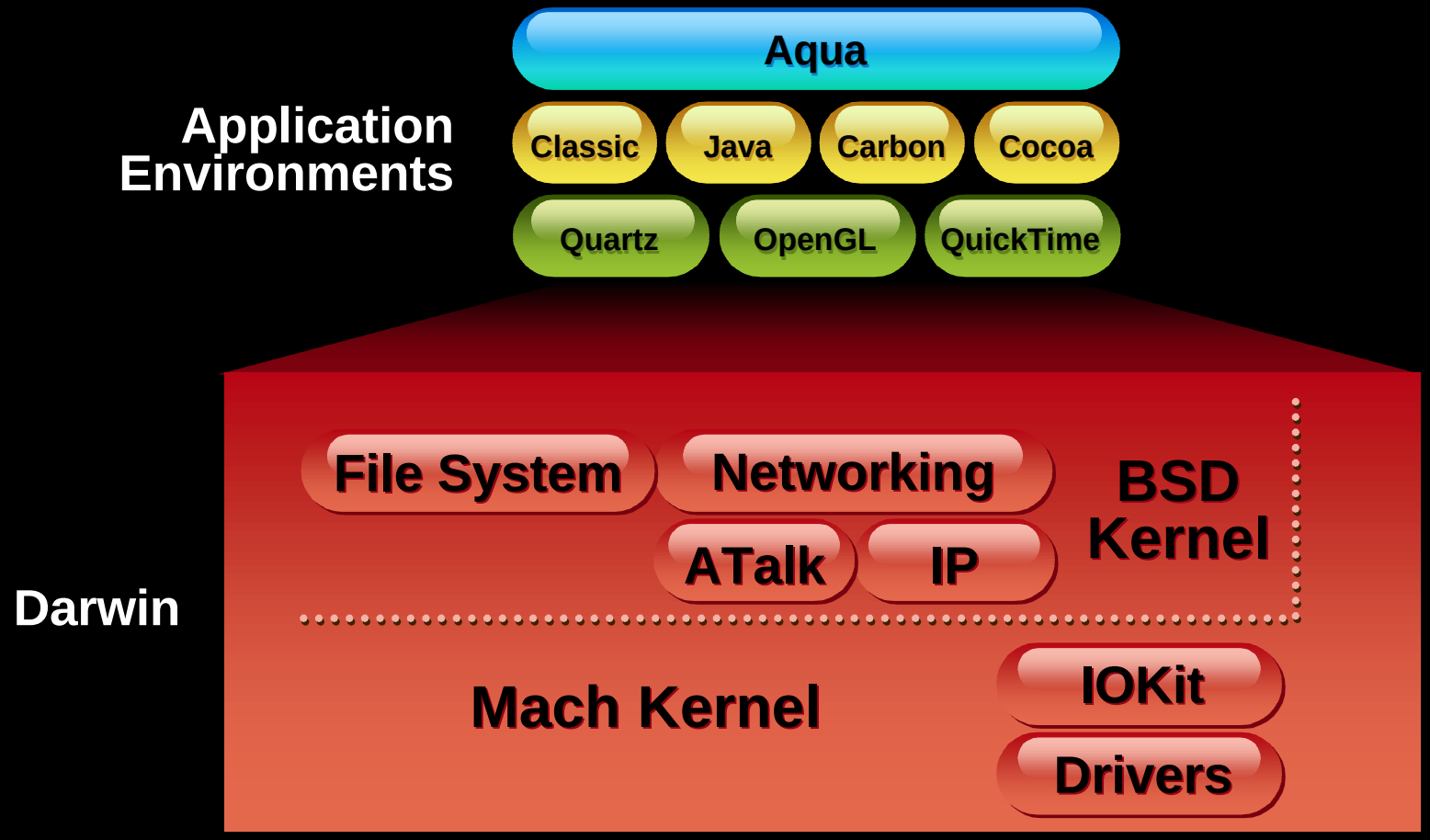
Mac OS X Model



Mac OS X Model: Darwin



Darwin/Core OS



What Is I/O Kit?

- To Driver developers
  - Makes it as easy as possible to develop correct Mac OS X drivers
- To Application developers
  - Provides discovery and rendezvous mechanism, and device abstractions
- To the OS
  - Provides an abstraction for each attached device

What Is I/O Kit?

- Framework and collection of Families for developing Mac OS X Drivers
- Delivered on the Mac OS X Developer CD
- Periodic Updates on ADR Website

What Is I/O Kit?

- Part of the Darwin kernel
- In-Kernel APIs presented in the Kernel.framework
- Application APIs presented in the IOKit.framework
- Object-Oriented Design (C++)

When to Use I/O Kit?

- Development of drivers that must run inside the Mac OS X kernel
- Development of applications that directly access drivers
- Not all drivers in Mac OS X are I/O Kit drivers:
  - Cameras, scanners, printers, . . .

Design Goals

- Simplify the very complex
  - MP and Preemption
  - Multiple Address Spaces
  - Architecture Neutrality
- Drivers focus on mechanism
- Keep policy out of drivers
- 3rd-party developers can extend Mac OS X just like Apple

I/O Kit Families and Drivers

- I/O Kit is just a framework
- No intrinsic protocol support
- Protocol-specifics are in Families
- Device-specifics are in Drivers

I/O Kit Families

- Families contain classes that present an abstract view of a type of device
- Families present abstractions upwards via “nub” objects
- Drivers inherit device-independent details by subclassing from Families

Common I/O Kit Families

- IOUSBFamily
- IOPCIFamily
- IOFireWireFamily
- IOSerialFamily
- IOSCSIFamily
- IOHIDFamily
- IONetworkFamily
- IOPCCardFamily

I/O Kit Drivers

- An I/O Kit driver is a “Leaf Class”
- Inherits from a Family
- Focuses on details of driving one specific device

The IORegistry

- Only in memory; never archived
- Is the set of active drivers
- Basically tree structured
- Tracks all relationships between drivers
- Contains Properties for each driver
- IORegistryExplorer.app, /usr/sbin/ioreg

Driver Matching

- Provided by Family
- Stages:
  - Device Discovery
  - Class Matching
  - Passive Matching
  - Active Matching
- Uses scores to break ties
- Very powerful and flexible

Matching Step 1: Device Discovery

- During boot, the PCI Host Bridge driver discovers a PCI Device
- It creates and attaches an IOPCIDevice object (nub)
- The new nub registers for matching

Matching Step 2: Class Matching

- I/O Kit generates a list of candidate drivers
- All known KEXTs are considered
- Those with IOKitPersonalities indicating an IOProviderClass compatible with the nub are included
- List is sorted in IOProbeScore order

Matching Step 3: Passive Matching

- Family-specific
- The personality dictionary for each driver is examined by the nub
- Inappropriate drivers are discarded
- Scores may be adjusted and the list resorted

Matching Step 4: Active Matching

- Each remaining driver is loaded and probe()d, in order
- Drivers that fail their probe are unloaded
- Remaining drivers are start()ed in probe score order until one succeeds (wins)

Developing Inside the Kernel

- Must use our tools (for now)
- Two-machine debugging
- Restricted language features
- Limited user interaction
- Limited libraries available (minimal libc)
- Resources are more costly
- Failures are fatal

Kernel Extensions

- A code-fragment container for kernel binaries
  - I/O Kit Drivers
  - Network Kernel Extensions (NKEs)
  - Filesystem Plug-ins
- Extension ≠ Mac OS X KEXT
- Only if you have to…

Kernel Extension Libraries

- Identified by Bundle Identifier and Version
  - com.apple.iokit.IOPCIFamily 1.0.0
- Dependency indicated by OSBundleLibraries property
- You should list all Libraries which export API you directly access

Kernel Extension Libraries

- You should list all Libraries which export API you directly access:
  - com.apple.kernel
  - com.apple.kernel.mach
  - com.apple.kernel.bsd
  - com.apple.kernel.iokit
  - com.apple.kernel.libkern
  - com.apple.iokit.IOPCIFamily
  - com.apple.iokit.IOUSBFamily

    . . .

KEXT Example: Structure

- IOUSBFamily.kext
  - Contents
    - Info.plist
    - MacOS
      - IOUSBFamily
    - PlugIns
      - AppleUSBComposite.kext
      - IOUSBUserClient.kext
      - IOUSBLib.bundle

…

KEXT Example: Structure

- IOUSBFamily.kext
  - Contents
    - Info.plist
    - MacOS
      - IOUSBFamily
    - PlugIns
      - AppleUSBComposite.kext
      - IOUSBUserClient.kext
      - IOUSBLib.bundle

…

KEXT Example: Info.plist

- Info.plist is an xml property list that identifies the KEXT and its contents
- Critical properties:

  CFBundleIdentifier = “com.you.driver”

  CFBundleVersion = “1.2.3b4”

  OSBundleLibraries = “com.apple.io…”

  IOKitPersonalities = { … }

KEXT Example: Structure

- IOUSBFamily.kext
  - Contents
    - Info.plist
    - MacOS
      - IOUSBFamily
    - PlugIns
      - AppleUSBComposite.kext
      - IOUSBUserClient.kext
      - IOUSBLib.bundle

…

KEXT Example: Binary

- Optional
- Mach-O format
- Run "**strip -x**" before shipping
  - Strips debug information and internal symbols
  - Will reduce binary size 90%

KEXT Example: Structure

- IOUSBFamily.kext
  - Contents
    - Info.plist
    - MacOS
      - IOUSBFamily
    - PlugIns
      - AppleUSBComposite.kext
      - IOUSBUserClient.kext
      - IOUSBLib.bundle
      - …

KEXT Example: Plug-ins

- Optional
- Allows collection of drivers to be delivered as single KEXT
- Can contain:
  - Other KEXTs
  - Device Interface libs (CFBundles)
- Only one level of nesting supported

User/Kernel Boundary

- Drivers are inside the kernel
- Apps are outside the kernel
  - Accesses must cross this "User/Kernel Boundary"
  - In-kernel implementation derived from IOUserClient
  - App-level Library can use CFPlugIn Interface

User/Kernel Boundary

- Your code belongs outside the kernel unless one or more are true:
  - Must process interrupts
  - Must deliver services to in-kernel components
  - Service needed by many clients much of the time

Device Interface Model

- Provides App access to many devices
- Wraps Mach mechanisms
  - System Calls
  - Mach Traps
  - Mach IPC
  - Mach shared memory
- To driver, looks like an in-kernel client
- To app, looks like a CFPlugIn (Mach-O)

Device Interface Model

1) Find the device of interest
2) Request a connection
3) Access away
4) Release the device

Device Interface Step 1: Find the Device of Interest

- Include **IOKit/IOKitLib.h** and Family **.h**
- Call **IOServiceGetMatchingServices()**
  - Returns list of matching devices
- Use **IOIteratorNext()** to scan for device(s) of interest

Device Interface Step 2: Request a Session

- Include:
  - **IOKit/IOCFPlugIn.h** and Family **.h**
- Call **IOCreatePlugInInterfaceForService()**
  - Returns **kCFPlugInInterfaceID**
- Call **(*dev)->QueryInterface(dev,…)**
  - Returns interface to desired device

Device Interface Step 3: Perform I/O to Device

- Details are family-specific, usually

```
(*dev)->open(dev,…)
…
(*dev)->whatever(dev,…)
…
(*dev)->close(dev)
```

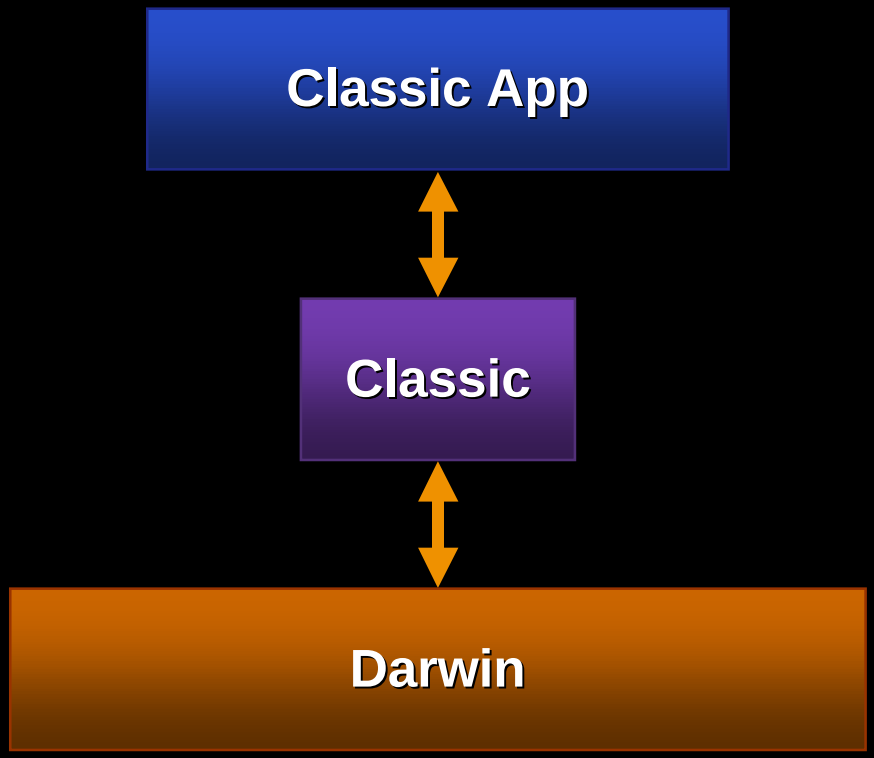
- Open/close used for arbitration

Device Interface Step 4: Release the Device

- Call **(*dev)->Release(dev)**
- See also “CFBundle & CFPlugIns” Release note

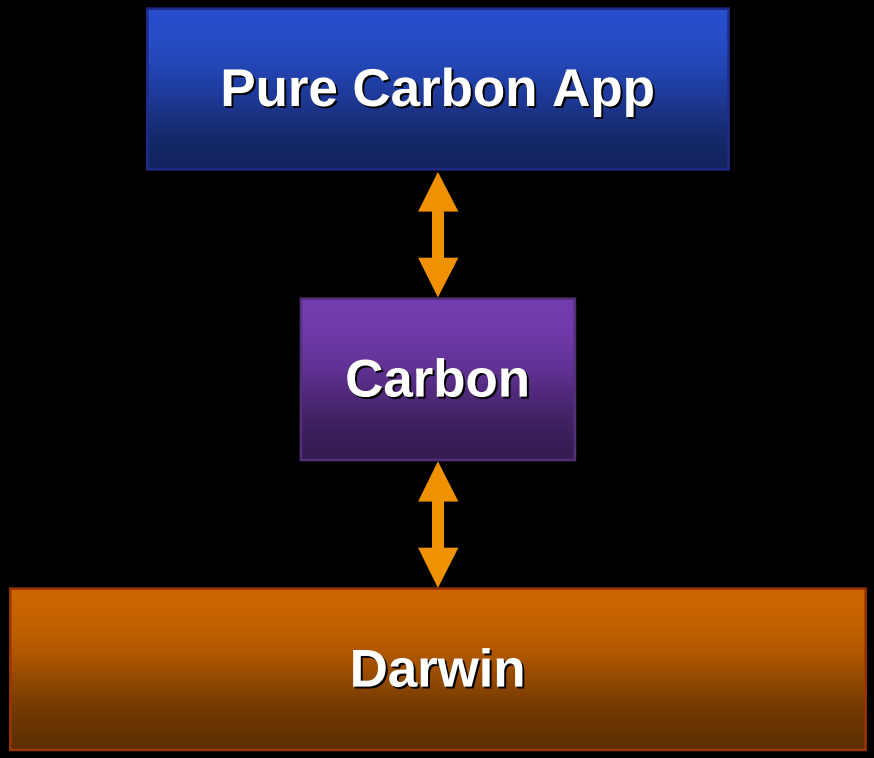
Driver Access from Classic Apps

All access goes through Classic APIs



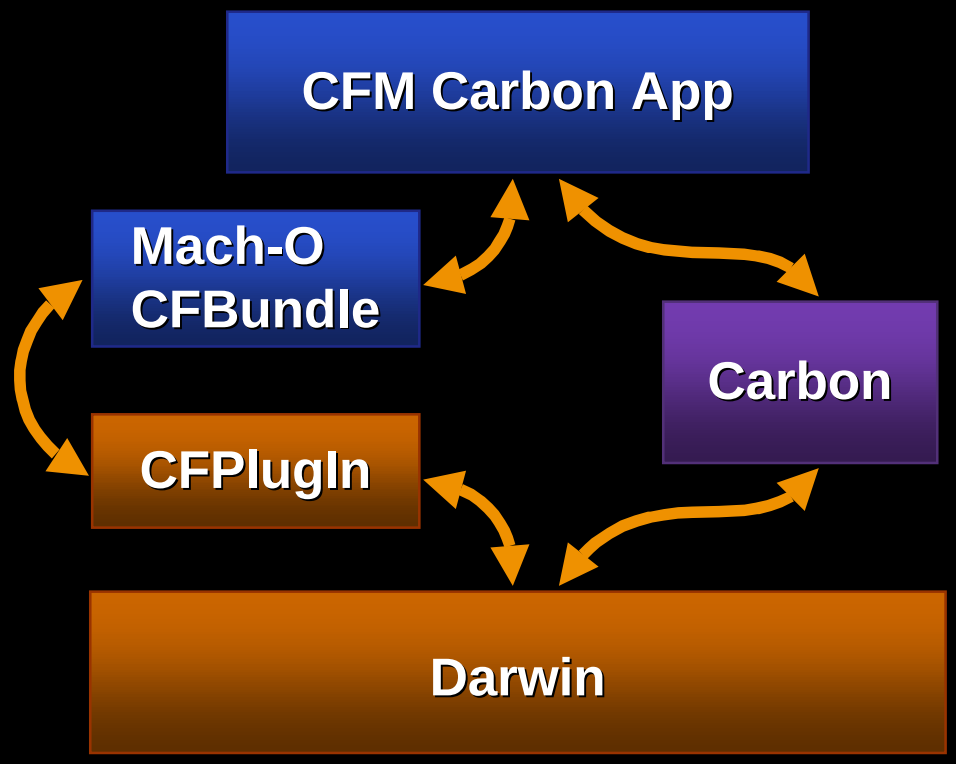
Access from Pure-Carbon Apps

All access goes through Carbon APIs



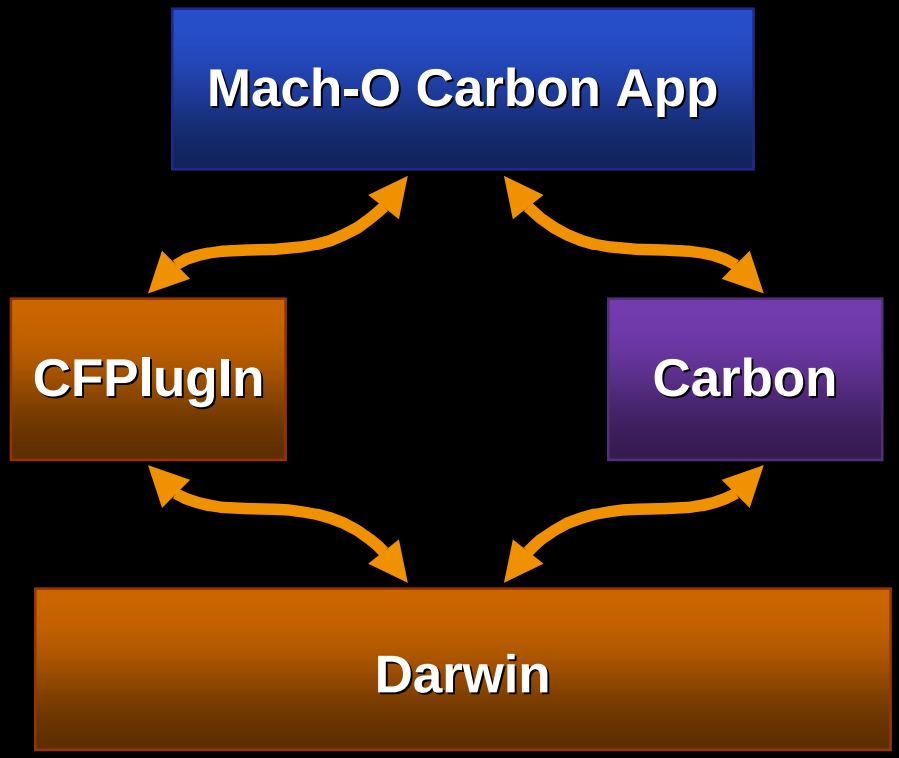
Access from CFM Carbon Apps

I/O Kit Native APIs available via CFPlugin



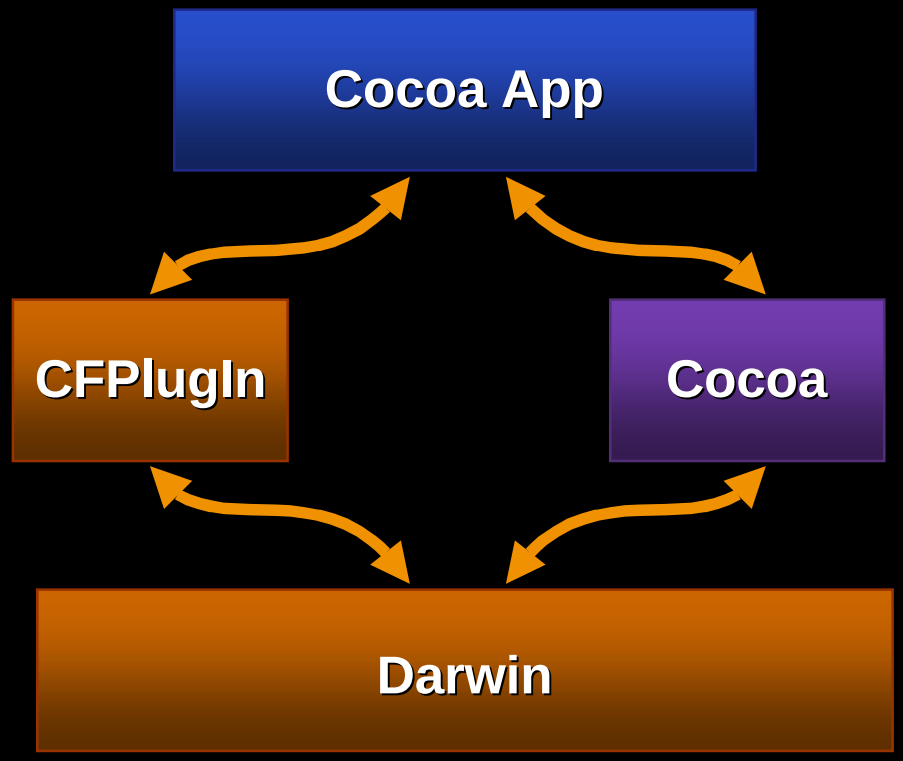
Access from Mach-O Carbon Apps

I/O Kit Native APIs directly available



Driver Access from Cocoa Apps

I/O Kit Native APIs directly available



Related Sessions

103 Darwin: The Foundation of Mac OS X	Room A1 **Mon., 2:00pm**
105 Graphics & Imaging: Overview of 2D and 3D Technologies	Room A1 **Mon., 5:00pm**
115 Open Source at Apple: Status and Plans for Apple's Open Sources	Room A2 **Wed., 10:30am**
117 Image Capture Framework: Image Capture Driver and App Development	Room C **Wed., 10:30am**

Related Sessions (Cont.)

135 Input for Games: HID Manager and Carbon Events	Room J2 **Fri., 10:30am**
140 The Darwin Kernel: Scheduling, Memory Management, etc.	Civic **Fri., 3:30pm**
202 FireWire Overview: Learn about Apple's FireWire Technology	Room A2 **Tue., 2:00pm**
203 USB Overview: Learn the Mac OS X USB Architecture	Room A2 **Tue., 3:30pm**

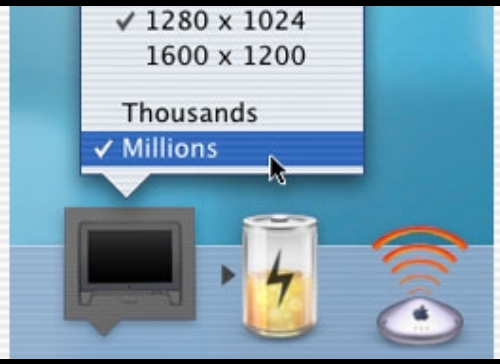
Related Sessions (Cont.)

204 I/O Kit: Storage Drivers: Mass Storage Architecture of Mac OS X	Room A2 **Tue., 5:00pm**
205 I/O Kit: PCI Drivers & Open Firmware PCI, AGP, NDRVs, KEXTs, Fcode, . . .	Room C **Wed., 9:00am**
206 FireWire In-Depth: Optimize your Mac OS X FW Drivers	Civic **Wed., 2:00pm**
207 USB In-Depth: Explore the Mac OS X USB Architecture	Civic **Wed., 3:30pm**

Who to Contact

Craig Keithley
FireWire & USB Technology Manager
Apple Worldwide Developer Relations
keithley@apple.com

Q&A



Dean Reece
I/O Kit Manager
dean.reece@apple.com

http://developer.apple.com/events/urls.html

WWDC

Worldwide Developers Conference 2001